Attorney Docket No.: 42P16557

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

VIRTUAL MANAGEMENT CONTROLLER TO COORDINATE PROCESSING BLADE MANAGEMENT IN A BLADE SERVER ENVIRONMENT

INVENTOR(S): **GUNDRALA D. GOUD**
VINCENT J. ZIMMER

PREPARED BY:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD, 7TH FLOOR
LOS ANGELES, CALIFORNIA 90025
(206) 292-8600

Express Mail Certificate of Mailing

"Express Mail" mailing label number: EV320118571US
Date of Deposit: September 24, 2003

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Mail Stop Patent Application, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Luci M. Arevalo
(Typed or printed name of person mailing paper or fee)

Luci M. Arevalo
(Signature of person mailing paper or fee)

September 24, 2003
(Date signed)

VIRTUAL MANAGEMENT CONTROLLER TO COORDINATE PROCESSING BLADE MANAGEMENT IN A BLADE SERVER ENVIRONMENT

TECHNICAL FIELD

[0001] This disclosure relates generally to processing blade management in a blade server environment, and in particular but not exclusively, relates to emulating a management controller of a processing blade with a software proxy layer.

BACKGROUND INFORMATION

[0002] Blade server architecture is an evolving technology that conveniently packages a server on a single board called a processing blade and houses a plurality of similar processing blades in a chassis. The processing blades can be easily installed or removed as desired and share common resources, such as input/output ("I/O" ports), a compact disc read only memory ("CD-ROM") drive, a floppy drive, a digital versatile disc ("DVD") drive, one or more network connections, etc. The ease with which the processing power can be scaled by adding or removing processing blades is a key feature driving the development and use of blade server technology. This scalable feature makes blade servers ideal candidates for performing tasks such as data and webpage hosting.

[0003] Processing blades typically are complete processing systems including one or more processors, firmware, system memory, one or more hard disks, one or more network interface cards ("NICs"), and the like. Known processing blades also generally include a baseboard management controller ("BMC") and an associated BMC firmware unit. A BMC is usually a coprocessor or service processor, separate from the one or

more main processors. The BMC communicates with a chassis management module ("CMM"), typically mounted on the chassis of the blade server, to coordinate and to manage operation of the individual processing blade.

[0004] In general, a blade server will house a plurality of processing blades, each having its own BMC and BMC firmware unit, which all coordinate with the CMM. The inclusion of the BMC and the BMC firmware unit in each processing blade substantially increases build of material ("BOM") costs. In addition, the BMCs and the BMC firmware units consume valuable real estate on the surface of the processing blades. These two adverse consequences of the BMCs and the BMC firmware units directly conflict with two key goals associated with the development and implementation of blade server technology—providing a high density and economical processing solution.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

[0006] FIG. 1 is a perspective view of a blade server including a plurality of processing blades.

[0007] FIG. 2 is a block diagram illustrating a known processing blade, including a baseboard management controller.

[0008] FIG. 3 is a block diagram illustrating how baseboard management controllers of known processing blades communicate with a chassis management module to coordinate access to shared resources.

[0009] FIG. 4 is block diagram illustrating a processing blade for executing a virtual management controller to facilitate communication with a chassis management module, in accordance with an embodiment of the present invention.

[0010] FIG. 5 is a block diagram illustrating how virtual management controllers executing on processing blades communicate with a chassis management module of a blade server to coordinate access to shared resources, in accordance with an embodiment of the present invention.

[0011] FIG. 6 is a flow chart illustrating a process for implementing the functionality of a baseboard management controller with a software proxy layer called a virtual management controller, in accordance with an embodiment of the present invention.

[0012] FIG. 7 is a block diagram illustrating linked software agents for implementing a virtual management controller, in accordance with an embodiment of the present invention.

[0013] FIG. 8 is a block diagram illustrating how applications/drivers executing in various different operating environments can communicate with a virtual management controller to communicate with a chassis management module of a blade server, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

[0014] Embodiments of a system and method for coordinating management of processing blades with a chassis management module of a blade server using software proxy layers are described herein. In the following description numerous specific details are set forth to provide a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that the invention can be practiced without one or more of the specific details, or with other methods, components, materials, etc. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of the invention.

[0015] Reference throughout this specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[0016] FIG. 1 is a perspective view of a blade server 100 including a chassis 105, processing blades 110, and a media tray 115. Chassis 105 includes a chassis management module ("CMM") 120 and a switch box 125. Media tray 115 optionally rests on top of chassis 105 and provides processing blades 110 with shared resources such as I/O ports (e.g., serial port, parallel port, universal serial bus port), I/O devices (e.g., monitor, keyboard, mouse), a CD-ROM drive 130, a floppy drive 135, and the like. Switch box 125 provides processing blades 110 with switchable access to a network 140

(e.g., local area network, wide area network, Internet). Typically, CMM 120 is an independent hardware module with dedicated firmware to perform management functions, such as for example, coordinating arbitration and allocation of shared resources between processing blades 110, logging system errors, coordinating fault resilient booting of processing blades 110, fan control, power supply monitoring and regulation, and the like.

[0017] FIG. 2 is a block diagram illustrating a known processing blade 110A, which may be installed into chassis 105. Processing blade 110A is a complete processing system including its own processor(s), firmware (e.g., basic input output system), system memory, hard disk(s), and network interface cards ("NICs"). Processing blade 110A is capable of booting and executing an operating system ("OS") over network 140. Processing blade 110A further includes a baseboard management controller ("BMC") 205 and associated BMC firmware unit 210. BMC 205 includes a service processor, which coordinates with CMM 120 to perform the management functions described above. For example, BMC firmware unit 210 contains firmware instructions executed by BMC 205 for processing blade 110A to request access to the shared resources provided by media tray 115. Communication between BMC 205 and CMM 120 occurs across a backplane (not shown) of chassis 105 where communication may occur out-of-band of communication across network 140. Similarly, each of the processing blades 110 can gain access to the shared resources provided by media tray 115 via the backplane of chassis 105.

[0018] FIG. 3 is a block diagram illustrating how each of processing blades 110 includes its own BMC 205 to communicate with CMM 120 to manage and to

coordinate processing blades 110. BMCs 205 are responsible for requesting permission from CMM 120 to access the shared resources. For example, CMM 120 arbitrates and allocates access to the shared resources of blade server 100 to ensure data collisions arising from bus contentions are avoided. To facilitate management of all processing blades 110, each processing blade 110 includes a BMC 205 to communication with CMM 120 via the backplane.

[0019] FIG. 4 is a block diagram illustrating a processing blade 400 for executing a virtual management controller ("VMC") 405 to facilitate communication with a chassis management module ("CMM") 410, in accordance with an embodiment of the present invention. The illustrated embodiment of processing blade 400 includes one or more processor(s) 415, a firmware unit 420, system memory 425, one or more hard disk(s) 430, communication links 435A and 435B, and a system bus 440.

[0020] The elements of processing blade 400 are interconnected as follows. Processor(s) 415 are communicatively coupled to firmware unit 420, system memory 425, hard disk(s) 430, and communication links 435A and 435B via system bus 440 to send and to received instructions thereto/therefrom. In one embodiment, firmware unit 420 is a flash memory device. In other embodiments, firmware unit 420 includes any one of read only memory ("ROM"), programmable ROM, erasable programmable ROM, electrically erasable programmable ROM, or the like. In one embodiment, system memory 425 includes random access memory ("RAM"). In one embodiment, communication links 435A and 435B include network interface cards ("NICs"). Hard disk(s) 430 may optionally include one or more of an integrated drive electronic ("IDE")

hard disk, an enhanced IDE ("EIDE") hard disk, a redundant array of independent disks ("RAID"), a small computer system interface ("SCSI") hard disk, and the like.

[0021] It should be appreciated that various other elements of processing blade 400 have been excluded from FIG. 4 and this discussion for the purposes of clarity. Furthermore, the illustrated embodiment of processing blade 400 is only one possible embodiment of a processing blade accordingly to the teachings of the present invention. One of ordinary skill in the art having the benefit of the present disclosure will understand various modifications to the architecture of processing blade 400 may be implemented within the scope of the present invention. For example, processing blade 400 may include only one communication link, instead of the two illustrated, or may include three or more to increase bandwidth capabilities of processing blade 400.

[0022] In one embodiment, firmware unit 420 stores a software entity or VMC 405 for communicating with CMM 410. VMC 405 acts as a software proxy layer to implement the functionality of a hardware BMC, such as BMC 205. Typically, VMC 405 is executed by one of processor(s) 415 to coordinate management functions of processing blade 400 with CMM 410 or to request access to shared resources, such as those provided by media tray 115. In one embodiment, VMC 405 is stored in firmware unit 420 and copied to system memory 425 to be executed therefrom by the selected one of processor(s) 415 during a management mode of operation of processing blade 400. In one embodiment, the management mode of operation is transparent to an operating system ("OS") runtime of processing blade 400.

[0023] FIG. 5 is a block diagram illustrating how a plurality of VMCs 405 executing on a plurality of processing blades 400 communicate with CMM 410 via a

backplane of a blade server chassis. Each VMC 405 is responsible to communicating with CMM 410 to coordinate management functions of the corresponding processing blade 400 and to gain access to the shared resources for its processing blade 400. Thus, embodiments of the present invention reduce the build of material ("BOM") costs of each processing blade 400 by eliminating the need for a hardware service processor and associated BMC firmware unit. Furthermore, processing blades 400 can be designed more compactly due to the surface area savings associated with elimination of a hardware service processor. Accordingly, embodiments of the present invention enable blade servers having a higher density of processing blades. Alternatively, the surface area saved by eliminating a hardware service processor could be occupied by an additional processor 415 or communication link 435, thereby increase processing power/bandwidth of processing blades 400.

[0024] FIG. 6 is a flow chart illustrating a process 600 for implementing the functionality of a BMC using VMC 405, in accordance with an embodiment of the present invention. In a process block 605, processing blade 400 is executing instructions during any one of a pre-boot runtime, an OS load sequence, and an OS runtime. In a decision block 610, a software entity executing on processing blade 400 determines that communication with CMM 410 is necessary to accomplish a desired task. The task may include requesting access to the shared resources (e.g., media tray 115, network 140 via switch box 125), reporting a system error (e.g., one or more of processor(s) 415 is hung/faulting, system memory 425 is corrupted, hard disk(s) 430 are corrupted or faulty, etc.) to CMM 410, coordinating with CMM 410 to perform fault resilient booting, event forwarding, and other tasks traditionally requiring the aid of BMC 205.

[0025] Referring to FIG. 7, a software entity 705 determines that communication with CMM 410 is necessary to accomplish the desired task. FIG. 7 is a block diagram illustrating linked software agents for implementing VMC 405 on processing blade 400, in accordance with an embodiment of the present invention. Software entity 705 represents any software entity executed by one of processor(s) 415. For example, software entity 705 may be a pre-boot device driver, an extensible firmware interface ("EFI") application, an OS driver, an OS application, or the like.

[0026] Returning to FIG. 6 and FIG. 7, in a process block 615, software entity 705 writes interrupt data to shared memory 710, as indicated by arrow 715. Shared memory 710 is a portion of system memory 425 which was allocated early in a boot-up process of processing blade 400 for storing the interrupt data. In one embodiment, the interrupt data written to shared memory 710 is data which indicates the desired task that software entity 705 wishes to accomplish.

[0027] In a process block 620, software entity 705 writes an interrupt value to an interrupt register 720, as indicated by arrow 725. In a process block 625, a synchronous management mode interrupt ("MMI") is generated in response to software entity 705 writing the interrupt value to interrupt register 720. The MMI is synchronous because it was initiated by a software entity, rather than an asynchronous hardware entity. It should be appreciated that embodiments of the present invention may include hardware entities generating an asynchronous MMI.

[0028] In a process block 630, processor(s) 415 activate a management mode of operation in response to the MMI. The management mode of operation is an execution mode of processor(s) 415 that is transparent to one or more of the pre-boot

runtime, the OS load sequence, and the OS runtime of processor(s) 415. Typically, the management mode of operation is transparent to all of the above. Activating the management mode of operation causes one of processor(s) 415 to jump to protected memory 730 and commence execution from a determined location therein. In one embodiment, protected memory 730 is a portion of system memory 425 that was allocated early in the boot-up process of processing blade 400 to establish a secure and stable execution environment for the management mode of operation.

[0029] In one embodiment where processor(s) 415 are 32-bit Intel Architecture ("IA-32") processors, the management mode of operation is a system management mode ("SMM"). SMM is specified by an *IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide* (2003) made available by Intel® Corporation. Since the 386SL processor was introduced by the Intel® Corporation, SMM has been available on 32-bit Intel Architecture ("IA-32") processors as an operation mode hidden to operating systems that executes code loaded by firmware. SMM is a special-purpose operating mode provided for handling system-wide functions like power management, system hardware control, or proprietary original equipment manufacturer ("OEM") designed code. The mode is deemed transparent or "hidden" because pre-boot applications, the OS, and OS runtime software applications cannot see it, or even access it. In this SMM embodiment, the MMI is referred to as a system management interrupt ("SMI") and protected memory 730 is referred to as system management random access memory ("SMRAM"). Only applications executed during SMM have access to SMRAM. When an event generates an SMI (e.g., software entity 705 writing the interrupt value to interrupt

register 720), processor(s) 415 respond by saving a substantial part of their current state in a state save map, initialize some registers to provide the SMM execution environment, and then begin execution inside SMM.

[0030] In one embodiment where processor(s) 415 are members of the Itanium® Processor Family ("IPF"), the management mode of operation is a platform management mode ("PMM"). PMM is roughly analogous to SMM and is activated upon processor(s) 415 receiving a MMI referred to as a platform management interrupt ("PMI").

[0031] Returning to FIG. 6, in a decision block 635 it is determine whether processing blade 400 includes more than one processor 415. If processing blade 400 includes multiple processor(s) 415, the processors are allocated to one of two groups—application processors ("APs") and a boot-strap processor ("BSP"). Although processing blade 400 may include several AP processors, only one of processor(s) 415 is a BSP at any given time. The current BSP is responsible for servicing the MMI. If processing blade 400 includes multiple processors 415, then process 600 continues to a process block 640.

[0032] In a process block 640, all of processors 415 are synchronized and the one of processors 415 currently designated as the BSP takes control and services the MMI as described below. Subsequently, process 600 continues to a process block 645. If processing blade 400 includes only a single processor 415, process 600 proceeds directly to process block 645 from decision block 635, without need to synchronize multiple processors 415.

[0033] As briefly mentioned above, upon activation of the management mode of operation (e.g., SMM, PMM, or the like) the BSP processor branches into protected memory 730 and executes an MMI handler 735. In process block 645, MMI handler 735 reads the interrupt value written to interrupt register 720 by software entity 705. As mentioned above, the interrupt value provides MMI handler 735 with information to determine why the management mode of operation was activated and to which address location within system memory 425 to branch to service the MMI. In one embodiment, MMI handler 735 includes a case statement correlating interrupt values to branch addresses.

[0034] In a decision block 650, MMI handler 735 determines whether the MMI was activated to invoke VMC 405, based on the interrupt value read from interrupt register 720. If the interrupt value corresponds to VMC 405, indicating that software entity 705 desires to invoke VMC 405 to interact with CMM 410, then the BSP processor branches to VMC 405, as indicated by an arrow 740, and process 600 continues to a process block 655.

[0035] In a process block 655, VMC 405 parses shared memory 710 to determine what task software entity 705 desires to accomplish and thereby determine its appropriate course of action. For example, if software entity 705 desires access to CD-ROM drive 130 of media tray 115, VMC 405 will parse the interrupt data written to shared memory 710 by software entity 705 to determine that software entity 705 desires access to CD-ROM drive 130. Once this is determined, VMC 405 can communicate with CMM 410 to arbitrate for access to CD-ROM drive 130.

[0036] In a process block 660, VMC 405 generates one or more command packets to transmit to CMM 410 based on the parsed interrupt data. The command packet(s) contain the requisite requests/information to enable software entity 705 to accomplish the desired task. In a decision block 665, VMC 405 waits until CMM 410 is available. If CMM 410 is currently busy performing other management tasks, VMC 405 manages the timeout in a process block 667. Managing a timeout may include simply waiting a predetermine period of time and then checking to see if CMM 405 is available or other known techniques for managing timeouts.

[0037] Once CMM 410 is available for communication with VMC 405, VMC 405 transmits the command packet(s) to CMM 410, in a process block 670. In a decision block 675 and a process block 677, VMC 405 waits for a response from CMM 410 and manages an timeouts that may occur, as described above. In a process block 680, CMM 410 returns results to VMC 405 in response to the command packet(s) transmitted in process block 670. The returned results may simply indicate that software entity 705 has been granted access to the requested shared resource, may indicate that software entity 705 has been denied access to the requested shared resources, may include an error code or may be other instructions, commands, or information from CMM 410 relating to any one of the management tasks performed by CMM 410.

[0038] In a process block 685, VMC 405 updates shared memory 710 with the results returned from CMM 410. Updating shared memory 710 may include copying the results into a designated portion of shared memory 710, overwriting the interrupt data written into shared memory 710 by software entity 705, parsing the results and storing

the parsed results to shared memory 710 in a predetermined and meaningful manner or any other sensible manner.

[0039] In a process block 690, VMC 405 clears interrupt register 720 or returns it to a default value. Although process 600 illustrates process block 690 occurring subsequently to VMC 405 updating shared memory 710, it should be appreciated that other embodiment of the present invention may include clearing interrupt register 720 prior to or concurrently with updating shared memory 710.

[0040] Once shared memory 710 has been updated and interrupt register 720 cleared, process 600 continues to a process block 695. In process block 695, VMC 405 branches back to MMI handler 735. If no other cases statements are satisfied within MMI handler 735 (i.e., MMI handler 735 determines there are no other tasks to execute during the instant management mode of operation), MMI handler 735 deactivates the management mode of operation and returns processor(s) 415 to their previous execution prior to activating the management mode of operation (return to process block 605). Thus, software entity 205 and other applications/drivers executing during the pre-boot runtime, the OS load sequence, or the OS runtime are unaware that the management mode of operation was activated and deactivated. In this manner, the management mode of operation is transparent to these operation modes.

[0041] FIG. 8 is block diagram illustrating how applications/drivers executing in various different operating environment can communicate with VMC 405 to in turn communicate with CMM 410, in accordance with an embodiment of the present invention. For example, applications and/or drivers executing in each of the following environments: processor abstraction layer ("PAL") 805, system abstraction layer – A

("SAL-A") 810 , a system abstraction layer – B ("SAL-B"), BIOS 820, extensible firmware interface ("EFI") 825, and an OS 830 can interact with VMC 405 by writing interrupt data to shared memory 710 and setting interrupt register 720 appropriately. In comparison, prior art methods required each environment to include independent software protocols to talk with BMC 205. Thus, embodiments of the present invention simplify the management and operation of processing blades within a blade server environment by eliminating the need to write redundant BMC protocols for each operating environment of processor(s) 415.

[0042] The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes, various equivalent modifications are possible within the scope of the invention, as those skilled in the relevant art will recognize.

[0043] These modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.